

Vermont Despreader / Correlator Specification

Approval

| Who | Group | Date Approved |
|----------------|------------|---------------|
| Gary Lai | Originator | |
| Paige Kolze | Hardware | |
| Hung Nguyen | Hardware | |
| Brad Taylor | Systems | |
| Jack Greenbaum | Software | |
| Alan Wessel | Marketing | |
| | | |

Revision History

| Revision | Date | Author | Description |
|----------|---------|----------|---|
| 1.0 | 4/18/01 | Gary Lai | Original (copied from desp_corr_v1.4.doc) |
| | | | |
| | | | |

09960710-091901
T06T60-0T209660

8x Despreader/Correlator Enhanced Multiplier Opcode

Conventions:

- Naming: 1-bit values, both real and imaginary one-bit values are referred to as codes, and sometimes as PN codes. The real part and imaginary parts of a complex value is known as {real-part, imaginary part}, {real, img}, {I,j}, and {I,Q}; with a preference (real:img) and (I,Q)
- Code mapping: we will adopt the convention where 0->1 and 1->-1. This allows us to treat the one bit code values as signs of 1 bit integers. This is compliant with CDMA2000 but contrary to some common usage. An XOR can be used to convert from this mapping to the opposite.
 - Example the code 01 implies 0 for the real part and 1 for the imaginary part
- Real-imag ordering: we will adopt the convention that the img part of a complex number is allocated to the LSB or little endian position. The motivation for this is to allow real on the left and img on the right when viewing 32-bit values as hex or binary displays
 - Example the code 01 encodes 1-j; assuming img or 'j' is in lsb
- earliest-latest ordering: We will adopt the convention that earliest samples in time are assigned LSB slots. This is in line with naming samples in ascending order when written in time sequence.
Example a time sequence of values on a port appears as - D0:D1:D2:D3
- 1-bit * 8-bit complex multiplier format: We will adopt the convention that we implement a mathematical complex multiply assuming the input sample are preformatted as real+img, real-imag pairs. Other conventions such real multiplies, and complex conjugates of the imaginary part of the code will require additional preformatting of the input data, but may be implemented as opcode options.
 - Example: $\{0,0\} * \{i,q\} = \{(I-q), (I+q)\}$;

Additions:

1-bit * 8-bit complex vector dot product definition: $SUM(code[] * data[])$

```
complex_1 code[8];
complex_8 data[8];
complex_8 dotproduct;
for(n=0;n<nelm;n++)
{
    dotproduct += code[n] * data[n];
}
```

2- CODE code encoding

| 1-bit complex integer format – CODE.q, CODE.q | |
|---|-------------------|
| CODE code value | Numerical meaning |
| 0 | +1.0 |
| 1 | -1.0 |

3 CODE format

| 1-bit complex integer format – CODE[1:0] | |
|--|------------------------|
| Bit | Numerical meaning |
| CODE[1] | I, 1, or real part |
| CODE[0] | Q, j or imaginary part |

4 Complex 16-bit data input format



Vermont Despreader / Correlator Specification

| |
|-------------------------|
| Document Control No. |
| 01-003 |

Revision 1.0

| 16-bit complex integer format – IN[31:0] | |
|--|---------------------|
| Field | Numerical meaning |
| IN[31:16] | I or real part |
| IN[15:00] | Q or imaginary part |

5- Complex 8-bit data input format0 16-bit aligned

| 8-bit complex integer format – IN[31:0] | |
|---|---------------------|
| Field | Numerical meaning |
| IN[23:16] | I or real part |
| IN[07:00] | q or imaginary part |

6- Dual Complex 8-bit data input format1 16-bit aligned

| Dual 8-bit complex integer format – IN[31:0] | |
|--|--------------------------------|
| Field | Numerical meaning |
| IN[31:24] | I1 or real part sample 1 |
| IN[15:08] | q1 or imaginary part, sample 1 |
| IN[23:16] | I0 or real part, sample 0 |
| IN[07:00] | q0 or imaginary part sample 0 |

0960710-091501

CODE formats

Each CODE bit pair is used to drive a mux/negate unit connect to two input bytes

I# refers to one of 4 input muxes, D# refers to registers in the delay chain

| OPCODE | 4XDESP8 | 8XDESP8 | 16XCorrelate |
|-------------------------------|------------------------|------------------------|--------------------------|
| Packing in 32- bit word | 0: i0 : 0 : q0 | i1 : i0 : q1 : q0 | 0: i0 : 0 : q0... |
| CODE bit | Data input | Data input | Data input |
| 0,1 | I0 [23:16], I0 [07:00] | I0 [23:16], I0 [07:00] | I0 [15:08], I0 [07:00] |
| 2,3 | I1 [23:16], I1 [07:00] | I0 [31:24], I0 [15:08] | D0 [15:08], D0 [07:00] |
| 4,5 | I2 [23:16], I2 [07:00] | I1 [23:16], I1 [07:00] | D1 [15:08], D1 [07:00] |
| 6,7 | I3 [23:16], I3 [07:00] | I1 [31:24], I1 [15:08] | D2 [15:08], D2 [07:00] |
| 8,9 | | I2 [23:16], I2 [07:00] | D3 [15:08], D3 [07:00] |
| 10,11 | | I2 [31:24], I2 [15:08] | D4 [15:08], D4 [07:00] |
| 12,13 | | I3 [23:16], I3 [07:00] | D5 [15:08], D5 [07:00] |
| 14,15 | | I3 [31:24], I3 [15:08] | D6 [15:08], D6 [07:00] |
| 16,17 | I0 [23:16], I0 [07:00] | I0 [23:16], I0 [07:00] | D7 [15:08], D7 [07:00] |
| 18,19 | I1 [23:16], I1 [07:00] | I0 [31:24], I0 [15:08] | D8 [15:08], D8 [07:00] |
| 20,21 | I2 [23:16], I2 [07:00] | I1 [23:16], I1 [07:00] | D9 [15:08], D9 [07:00] |
| 22,23 | I3 [23:16], I3 [07:00] | I1 [31:24], I1 [15:08] | D10 [15:08], D10 [07:00] |
| 24,25 | | I2 [23:16], I2 [07:00] | D11 [15:08], D11 [07:00] |
| 26,27 | | I2 [31:24], I2 [15:08] | D12 [15:08], D12 [07:00] |
| 28,29 | | I3 [23:16], I3 [07:00] | D13 [15:08], D13 [07:00] |
| 30,31 | | I3 [31:24], I3 [15:08] | D14 [15:08], D14 [07:00] |

0960710-09101
T06T60-0T209660



CHAMELEON
SYSTEMS

Vermont Despreader / Correlator Specification

Document
Control No.
01-003

Revision 1.0

CODE multiply routing

A mux is employed to route CODE bits to the correct CODE multiply unit (the mux-negate unit) with the truth table below. The following table summarizes the routing to 32 mux negate units as a function of opcode. The 16XADD and 16XASUB opcodes could also be implemented in the mux/neg block instead of the mux

| CODE (real, img) | mapping | result.real | result.img |
|---------------------|---------|-------------|------------|
| 00 | +1, +1 | +r | +i |
| 01 | +1, -1 | +i | -r |
| 10 | -1, +1 | -i | +r |
| 11 | -1, 1 | -r | -i |

| OPCODE | Despreader | 4XDESP | 8XDESP | 16XCorrelate |
|-----------------------|------------|--------------|--------------|--------------|
| mux negate unit | | C src bit | C src bit | C src bit |
| x00 | T0.img | c[0,1] | c[0,1] | c[0,1] |
| x01 | T0.img | c[2,3] | c[4,5] | c[2,3] |
| x02 | T0.img | c[4,5] | c[8,9] | c[4,5] |
| x03 | T0.img | c[6,7] | c[12,13] | c[6,7] |
| x04 | T0.img | - | c[2,3] | c[8,9] |
| x05 | T0.img | - | c[6,7] | c[10,11] |
| x06 | T0.img | - | c[10,11] | c[12,13] |
| x07 | T0.img | - | c[14,15] | c[14,15] |
| y08 | T0.real | c[0,1] | c[0,1] | c[0,1] |
| y09 | T0.real | c[2,3] | c[4,5] | c[2,3] |
| y10 | T0.real | c[4,5] | c[8,9] | c[4,5] |
| y11 | T0.real | c[6,7] | c[12,13] | c[6,7] |
| y12 | T0.real | - | c[2,3] | c[8,9] |
| y13 | T0.real | - | c[6,7] | c[10,11] |
| y14 | T0.real | - | c[10,11] | c[12,13] |
| y15 | T0.real | - | c[14,15] | c[14,15] |
| x16 | T1.img | c[16,17] | c[16,17] | c[16,17] |
| x17 | T1.img | c[18,19] | c[20,21] | c[18,19] |
| x18 | T1.img | c[20,21] | c[24,25] | c[20,21] |
| x19 | T1.img | c[22,23] | c[28,29] | c[22,23] |
| x20 | T1.img | - | c[18,19] | c[24,25] |
| x21 | T1.img | - | c[22,23] | c[26,27] |
| x22 | T1.img | - | c[26,27] | c[28,29] |
| x23 | T1.img | - | c[30,31] | c[30,31] |
| y24 | T1.real | c[16,17] | c[16,17] | c[16,17] |
| y25 | T1.real | c[18,19] | c[20,21] | c[18,19] |
| y26 | T1.real | c[20,21] | c[24,25] | c[20,21] |
| y27 | T1.real | c[22,23] | c[28,29] | c[22,23] |
| y28 | T1.real | - | c[18,19] | c[24,25] |
| y29 | T1.real | - | c[22,23] | c[26,27] |
| y30 | T1.real | - | c[26,27] | c[28,29] |
| y31 | T1.real | - | c[30,31] | c[30,31] |

09560710-091901

Multiply Unit

CODE Multiply format

The code multiply unit multiplies 2 complex values, a 1-bit value, known as the code and an 8-bit complex pair known as data.

This leads to the table:

| CODE(real,img) | result.real= | result.img= |
|----------------|--------------------|--------------------|
| 00 -> 1, 1 | data.r - data.i; | data.r + data.q; |
| 01 -> 1, -1 | data.r + data.i; | - data.r + data.q; |
| 10 -> -1, 1 | - data.r - data.i; | data.r - data.q; |
| 11 -> -1, -1 | - data.r + data.i; | - data.r - data.q; |

For efficient implementation this is to be implemented in the despreader by:

- 1) requiring the data to be preformatted as: data.r = r-i, data.i=r+i;
- 2) using a mux followed by a negate to implement the multiply as follows:

| CODE(real,img) | result.real | result.img |
|----------------|-------------|------------|
| 00 -> 1, 1 | r - i | r + i |
| 01 -> 1, -1 | r + i | -(r - i) |
| 10 -> -1, 1 | -(r + i) | r - i |
| 11 -> -1, -1 | -(r - i) | -(r + i) |

If a 45 degree rotation and scaling is allowed as is ok when pilot and data are decoded together, the pre-formatting can be dropped to yield the following function table:

| CODE(real,img) | result.real | result.img |
|----------------|-------------|------------|
| 00 -> 1, 1 | r | i |
| 01 -> 1, -1 | i | -(r) |
| 10 -> -1, 1 | -(i) | r |
| 11 -> -1, -1 | -(r) | -(i) |

The real part is close to the UMTS encoding:

| bits | UMTS |
|------|------|
| 00 | + i |
| 01 | + r |
| 10 | - r |
| 11 | - i |

Mux-negate Options

Besides complex multiply other modes of the mux negate units are useful

These modes are

complex - the normal multiply

complex-conjugate - complement the imaginary part of the code before multiply

zero - force code to 00 to effect an adder

real - use the real part of the code to negate the real part of the data and the img part of the code to negate the img part of the data.

The following truth table would apply if we decided to implement these additional modes (assume data at mux is called real, img)

assuming the 4 modes are adopted, we can use input mux bits to for the source of the bits.

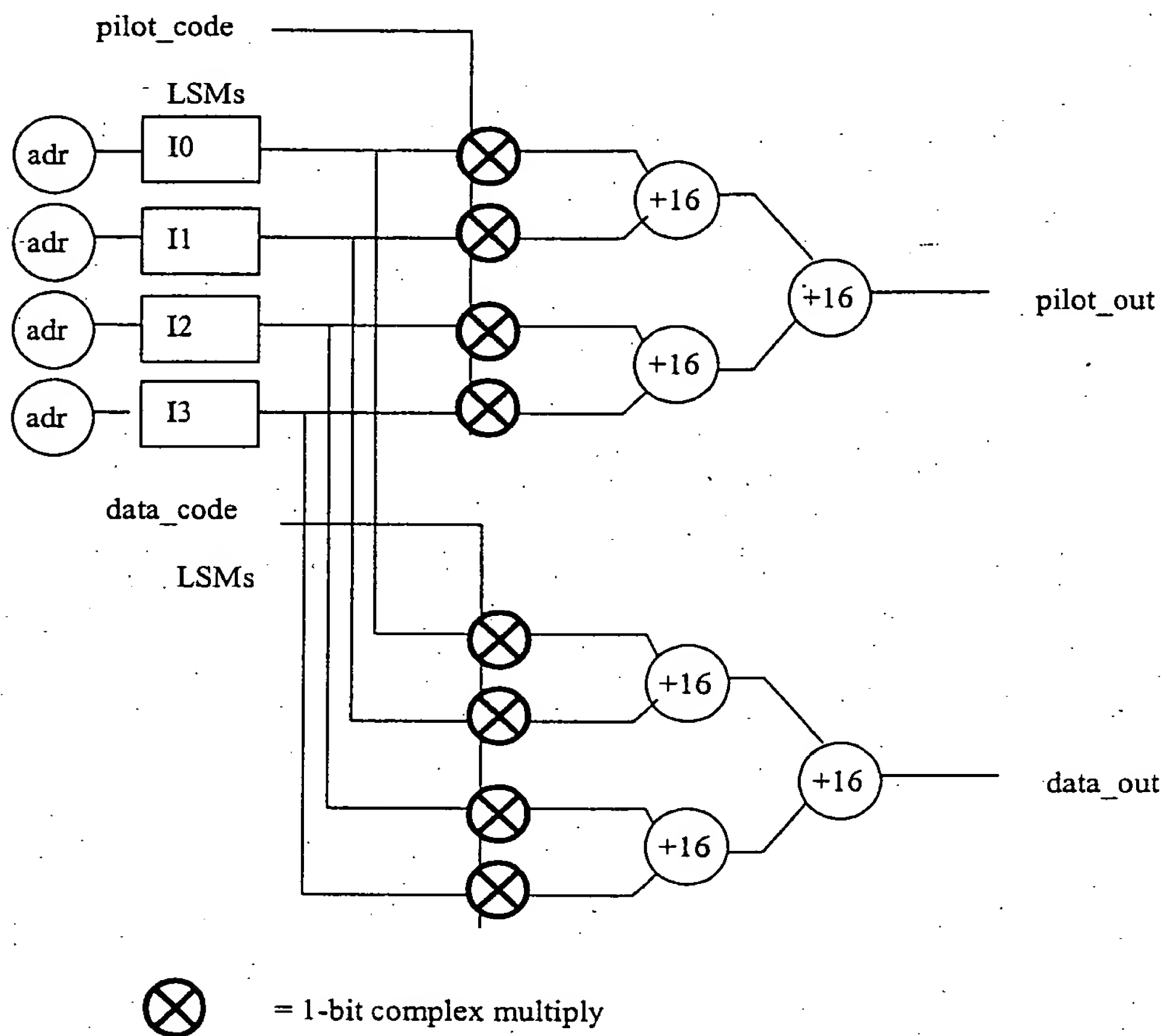
| mode | code | real result | img result |
|-------------|------|-------------|------------|
| complex | 00 | real | img |
| complex | 01 | img | -real |
| complex | 10 | -img | real |
| complex | 11 | -real | -img |
| complex-cnj | 01 | real | img |
| complex-cnj | 00 | img | -real |
| complex-cnj | 11 | -img | real |
| complex-cnj | 10 | -real | -img |
| real-r* | 0x | real | |
| real-r | 1x | -real | |
| real-i** | x0 | | img |
| real-i | x1 | | -img |
| zero | xx | real | img |

* real mode selects the real input and uses code[1] to control negation for the real output.

** real mode select the img input and uses code[0] to control negation for the img output.

Despreading

Despreading is used in rake receivers. The common case used in both 1XRTT and UMTS is to despread a symbol against 2 separate codes to recover pilot and data channels. The circuit below is used as a test case to evaluate despreading performance for a variety of architectures. It despreads 4 16-bit or 8 8-bit complex input samples known as "chips" to form two complex results corresponding to the pilot and data outputs of a rake desreader. Each input is stored as 8-bit complex data which may be unpacked to 16-bit complex data. The input data is assumed to be stored in separate LSM memories, and is addressed in such a fashion as to read out a contiguous neighborhood of 4 samples separated by a 1 chip time delay.



Vermont architecture enhancements which increase the number of chips per tile are:

- the address generator
- support for 8-bit unpacking
- support for addsub16 and subadd16 instructions.

Vermont plus architecture enhancements which increase the number of chips per tile are:

- Adder tree in 2X multipliers
- desreader opcode in enhanced Vermont

The data storage format for input data is important for efficiency. In some cases increased performance can be obtained if data is stored in memory as 16-bit data or in a redundant form of 8-bit data. The effect of various hw options has been summarized in table 1 and the implementations summarized in table 2.

Table 1 - Number of chips per tile for despreading data in memory against 2 CODE codes



CHAMELEON
SYSTEMS, LLC

Vermont Despreader / Correlator Specification

Document
Control No.
01-003

Revision 1.0

| function | data type | CS2112 | Vermont | Vermont 2Xmul | Vermont SBBA | Vermont EX |
|----------|-----------|--------|---------|------------------|-----------------|---------------|
| despread | 8-bit* | 1 | 1.4 | 2.3 | - | 4 |
| despread | 16-bit | 1 | 1.75 | 3.5 | - | 4 |
| despread | 2X8-bit** | 1.4 | - | - | 3.5 | 8 |
| corr | 8-bit | 52 | 64 | 64 | 64 | 192 |

* stores 8-bit complex data in memory as 32-bit word organized as:

+i:-q:+q:+i;

**stores 8-bit complex data in memory as 32-bit word organized as:

i0:q0:i1:q1,

i1:q1:i2:q2;

Table 2 - The table below details the DPU usage for despreading modes used above:

| format | chip | dpu0 | dpu1 | dpu2 | dpu3 | mult | nDPU pilot | nDPU data | nDPU tot | chip/ tile |
|--------|-----------|-----------------------|------------------|------|------|----------------------|---------------|--------------|-------------|---------------|
| 8-bit | 2112 | mem | unpack negate | swap | tree | | 4 | 3 | 7 | 1 |
| 8-bit | V | mem, unpack | negate swap | tree | | | 3 | 2 | 5 | 1.4 |
| 8-bit | V2x | mem unpack | negate swap | | | tree | 2 | 1 | 3 | 2.3 |
| 8-bit | Vex | mem unpack | | | | codemu lt tree | 1 | 0 | 1 | 4 |
| 16-bit | 2112 | mem | negate | swap | tree | tree | 4 | 3 | 7 | 1 |
| 16-bit | V | mem negate swap | tree | | | | 2 | 2 | 4 | 1.75 |
| 16-bit | V2x | mem negate swap | | | | tree | 1 | 1 | 2 | 3.5 |
| 16-bit | Vex | mem | | | | codemu lt tree | 1 | 0 | 1 | 4 |
| 2X8bit | 2112 | mem | negate swap | tree | | | 3 | 2 | 5 | 1.4 |
| 2X8bit | V sbba | mem sbba | | | | tree | 2 | 2 | 4 | 3.5 |
| 2X8bit | Vex | mem | | | | codemu lt tree | 1 | 0 | 1 | 8 |

1XRTT / UMTS Rake Receiver channel count

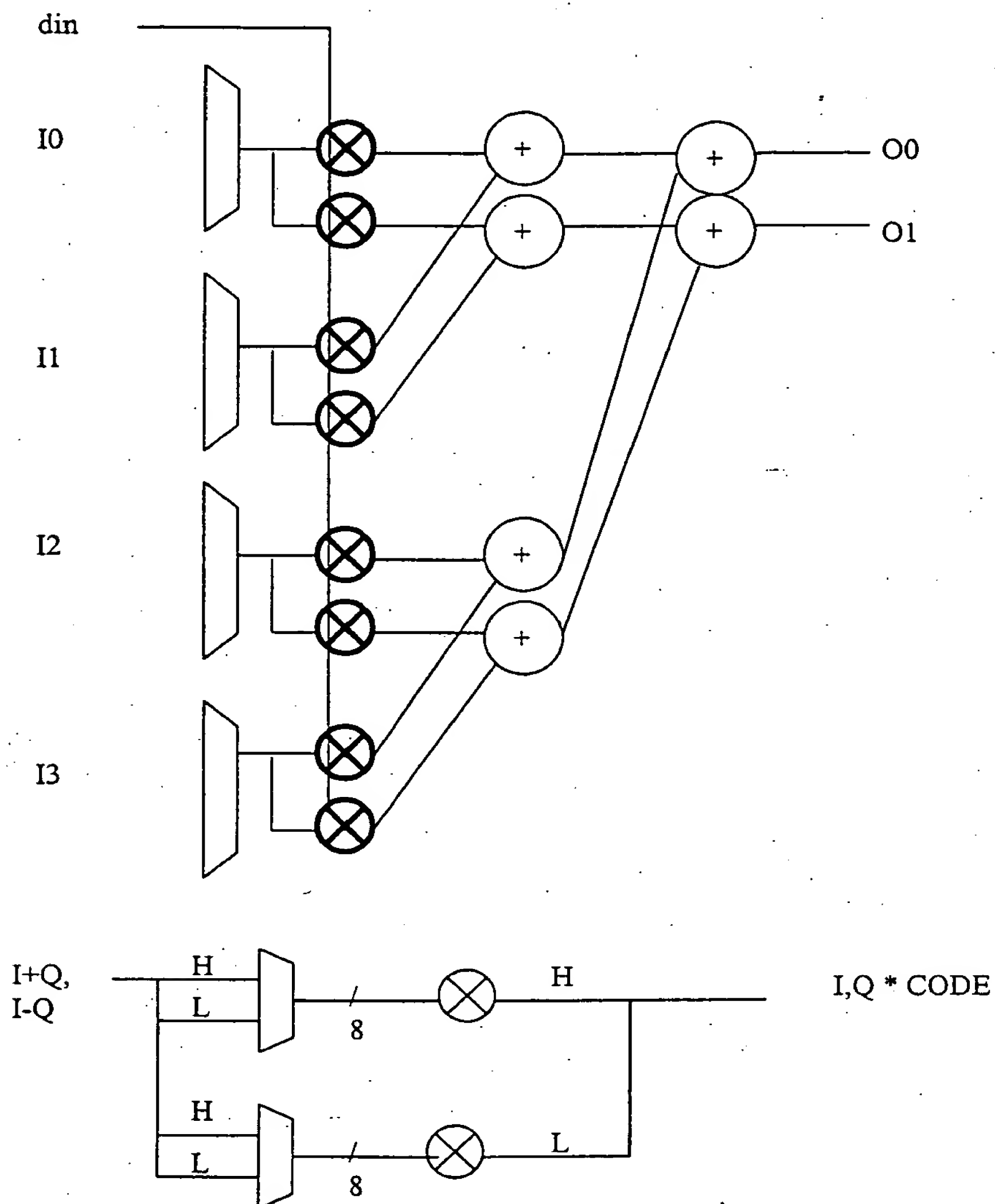
Based on the despreading performance and a 150 MHz clock for Vermont, the estimated 1XRTT and UMTS rake receiver channel count is summarized below :

| | CS2112 DPUs | CS2112 channels | Vermont channels | Vermont 2Xmul channels | Vermont EX channels |
|----------------|----------------|--------------------|---------------------|------------------------------|---------------------------|
| 1XRTT channels | 50 | 50 | 75 | 100 | 150-200 |
| UMTS channel | 32 | 16 | 24 | 32 | 32-48 |

09960710-091904
T06T60-0T209660

Despreading Implementation 1

The diagram below implements a 4 chip despreader to two different CODE codes



16-bit implementation of despreading opcode

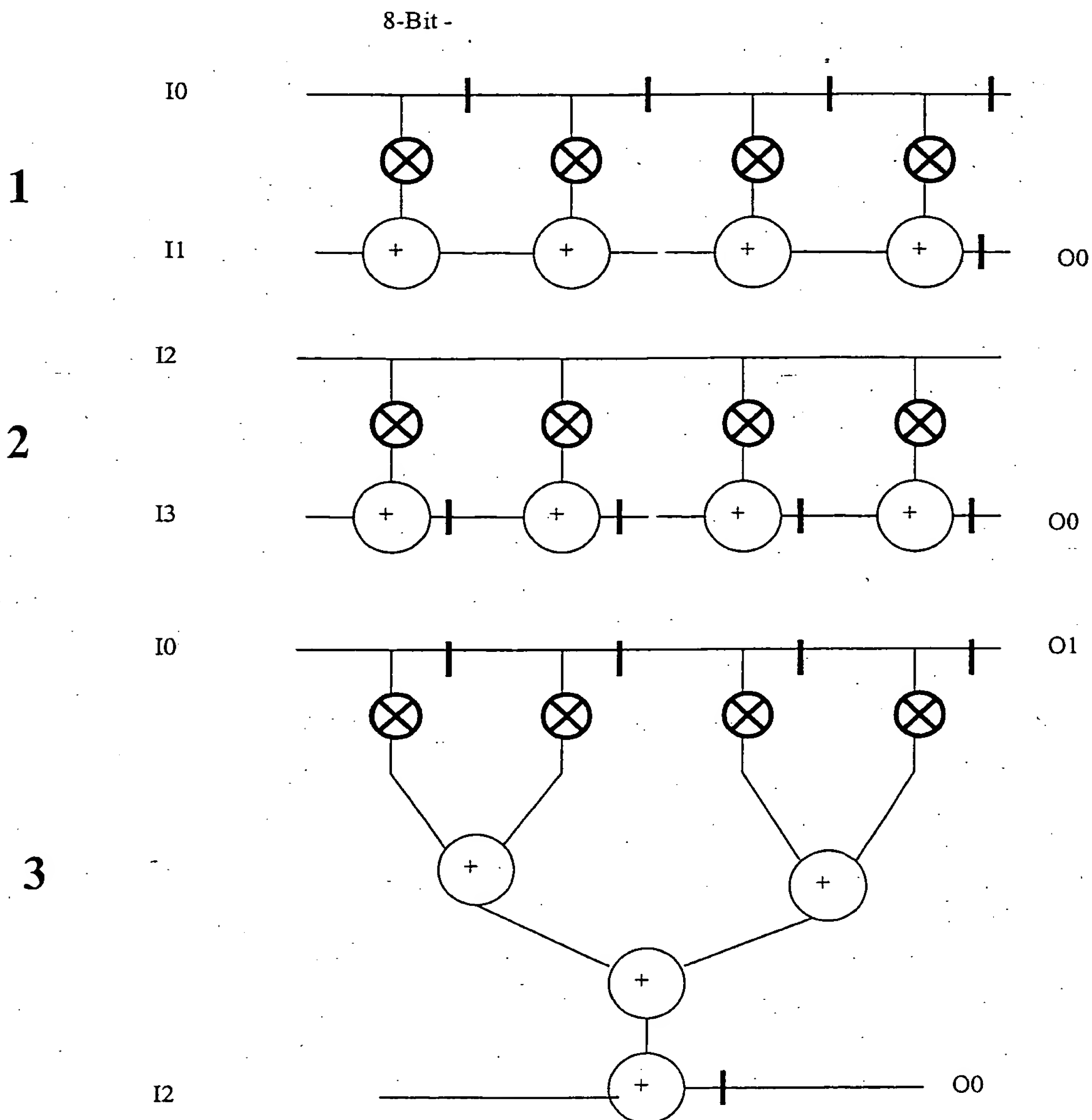
| CODE | O[31:16] = | O[15:0] = |
|------|-----------------|-----------------|
| 00 | $-H = -(I - Q)$ | $L = -(I + Q)$ |
| 01 | $-L = -(I + Q)$ | $H = (I - Q)$ |
| 10 | $L = (I + Q)$ | $-H = -(I - Q)$ |
| 11 | $H = (I - Q)$ | $L = (I + Q)$ |

CODE(real,img) result.real result.img

| | | |
|--------------|------------|------------|
| 00 -> -1, -1 | $-(r - i)$ | $-(r + i)$ |
| 01 -> -1, 1 | $-(r + i)$ | $r - i$ |
| 10 -> 1, -1 | $r + i$ | $-(r - i)$ |
| 11 -> 1, 1 | $r - i$ | $r + i$ |

Correlation circuit:

The following circuits implement the same correlation function:



Correlation circuits. Circuit 3 is implemented as the correlation opcode

Despreader Trees without input delay 3

A despreader tree can be constructed to implement dual 4-chip despreader for 16-bit data and a dual 8-chip despread for 8-bit data. 4 despread trees are needed, one for each 16-bit output field.

| Function | Output | Function |
|-------------------|-----------|---------------|
| Despreader Trees0 | O0[15:00] | real - i |
| Despreader Trees1 | O0[31:16] | imaginary - q |
| Despreader Trees2 | O1[15:00] | real - i |
| Despreader Trees3 | O1[31:16] | imaginary - q |

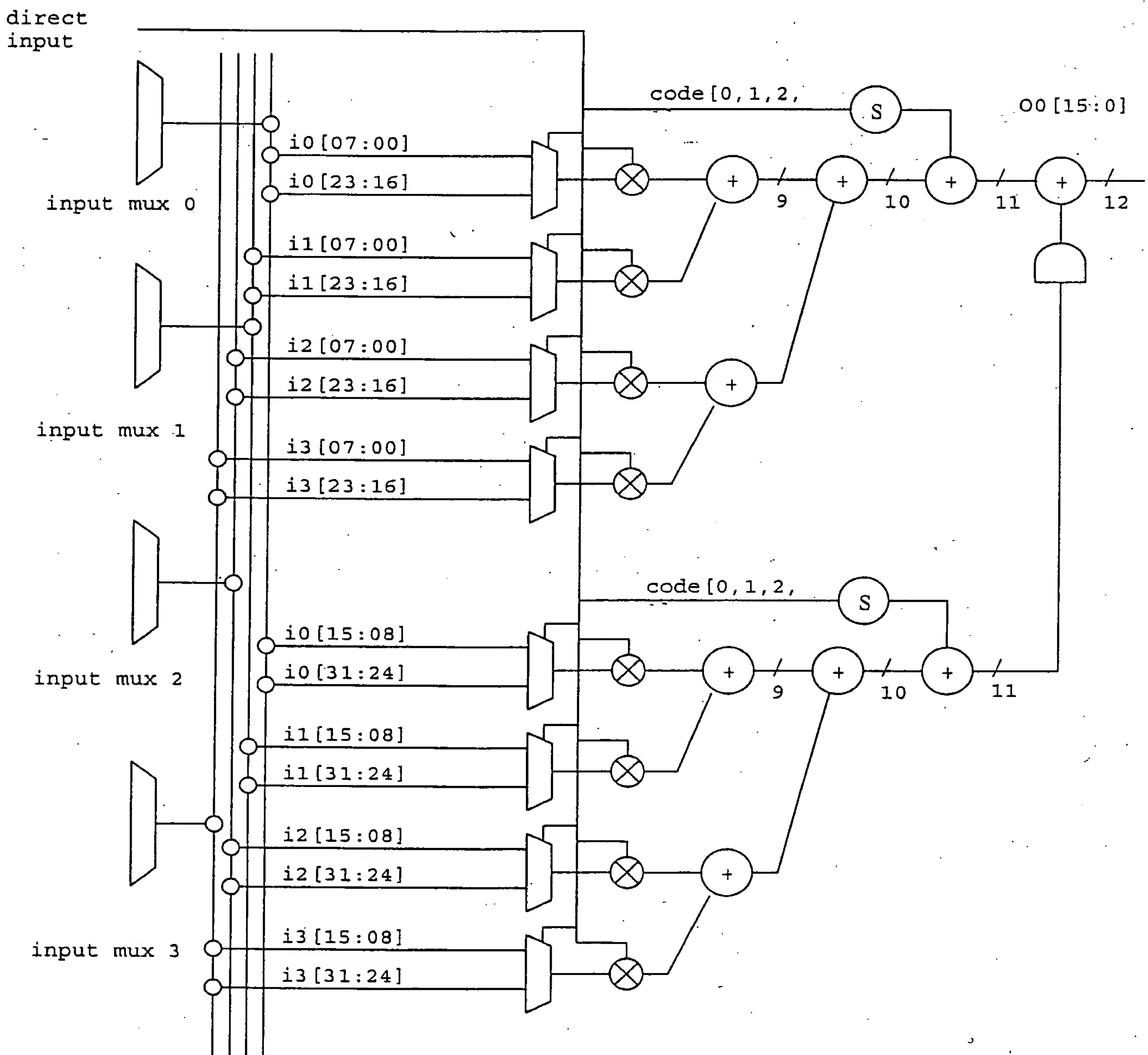


Fig 2 - Despread Tree 0



Vermont Despreader / Correlator Specification

| |
|-------------------------|
| Document Control No. |
| 01-003 |

Revision 1.0

Despreader Trees with input delay

Adding a separate input mux and delay chain enables a dual correlation function to be implemented with only one external DPU. For this mode Output O0 is the sum of C0+C1;

| Function | Function | Output - despread | Output - correlation | Chain input | Chain output |
|-------------------|---------------|----------------------|-------------------------|--------------------|--------------------|
| Despreader Trees0 | real - i | O0[15:00] | C0[15:00] | I0[23:16],I0[7:0] | chain[23:16],[7:0] |
| Despreader Trees1 | imaginary - q | O0[31:16] | C0[31:16] | | |
| Despreader Trees2 | real - i | O1[15:00] | C1[15:00] | chain[23:16],[7:0] | O1[15:00] |
| Despreader Trees3 | imaginary - q | O1[31:16] | C1[31:16] | | |

00960710-091901
106160"07209660

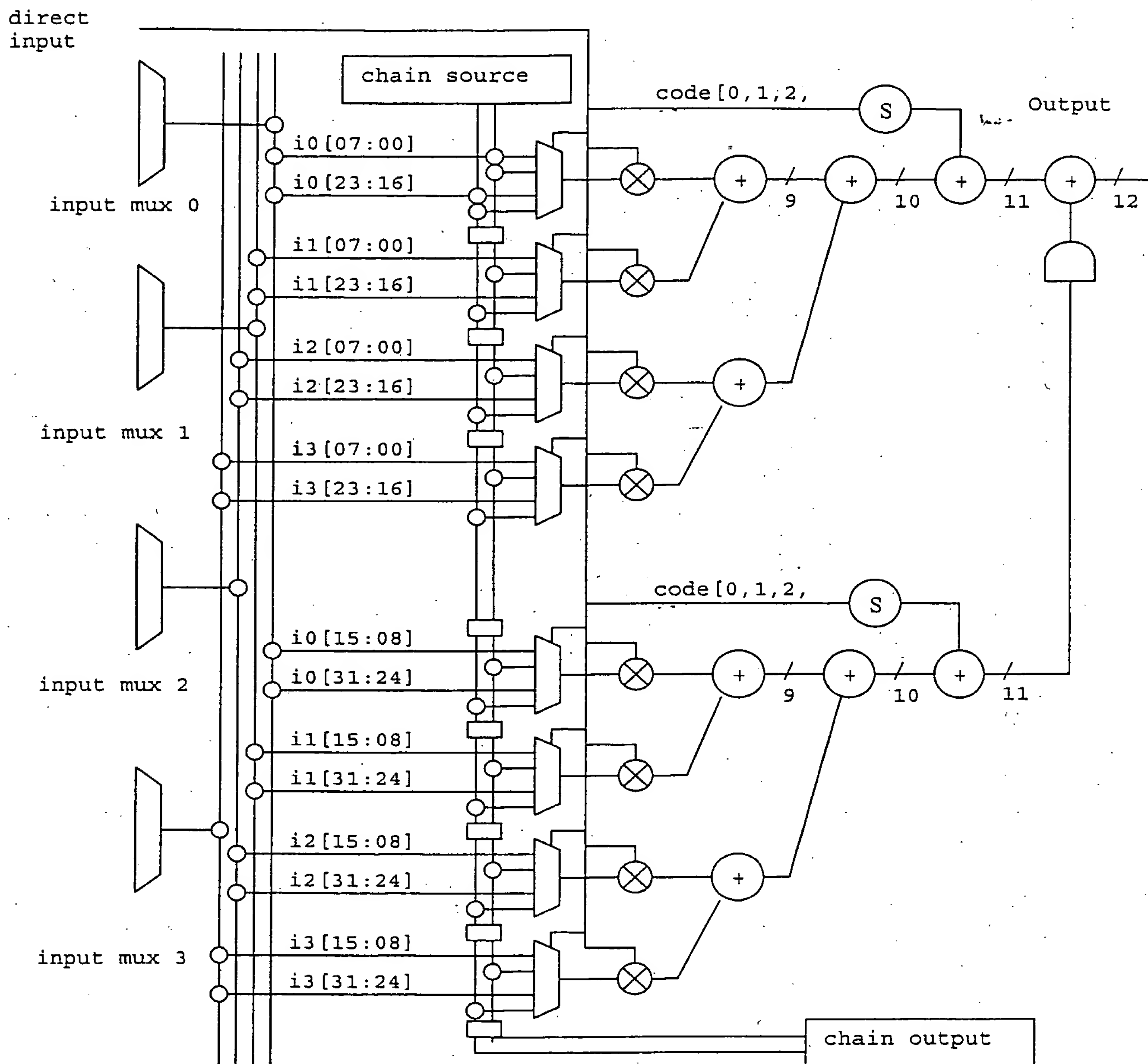


Fig 2 - Despread Tree 0



Vermont Despreader / Correlator Specification

| |
|-------------------------|
| Document Control No. |
| 01-003 |

Revision 1.0

Physical Layout

| | elements per output | total |
|----------------------------------|---------------------|-------|
| 4:1 Muxes 8-bits | 8 | 32 |
| pipe regs 8-bit | 16 | 64 |
| XOR 8-bit | 8 | 32 |
| adders 8-bit | 4 | 16 |
| adders 9-bit | 2 | 8 |
| adders 10-bit | 3 | 12 |
| Total blocks | 352 | 1408 |
| Total blocks in 4:1 mux and pipe | 192 | 768 |

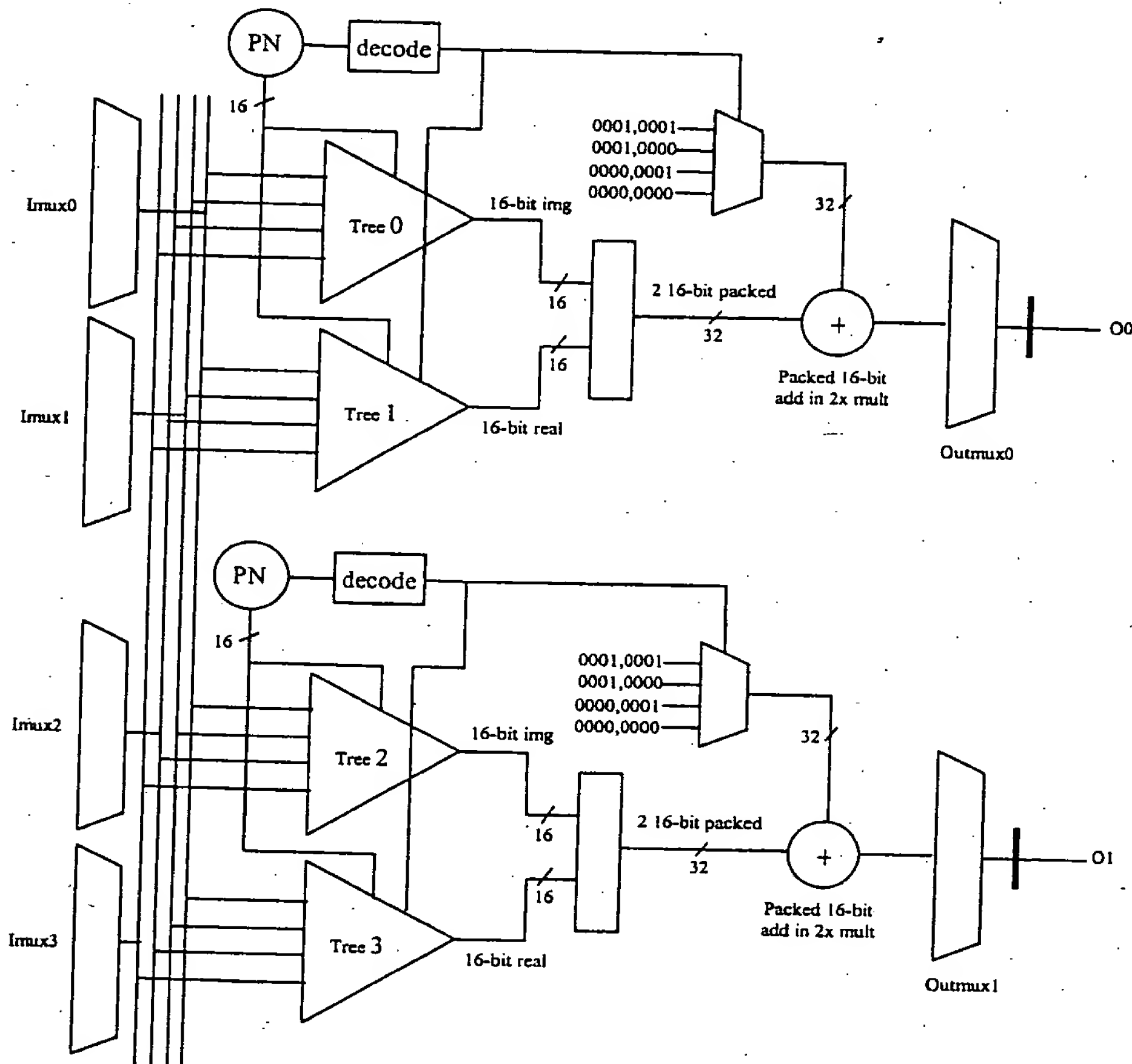
Loads per input = 4

$$1408 * 100 \text{ um sq} = .1408 \text{ mm sq}$$

| O1[31:16] | O0[31:16] | O1[31:16] | O0[31:16] |
|-----------|-----------|-----------|-----------|
| i0 | | | |
| i1 | | | |
| i2 | | | |
| i3 | | | |
| m00 | i01 | i02 | i01 |
| m01 | | | |
| a0 | a1 | a2 | a3 |
| a4 | a6 | a5 | a7 |
| m10 | | | |
| m11 | | | |
| a0 | a1 | a2 | a3 |
| a4 | a6 | a5 | a7 |
| m20 | | | |
| m21 | | | |
| a0 | a1 | a2 | a3 |
| a4 | a6 | a5 | a7 |
| m30 | | | |
| m31 | | | |
| a0 | a1 | a2 | a3 |
| a4 | a6 | a5 | a7 |

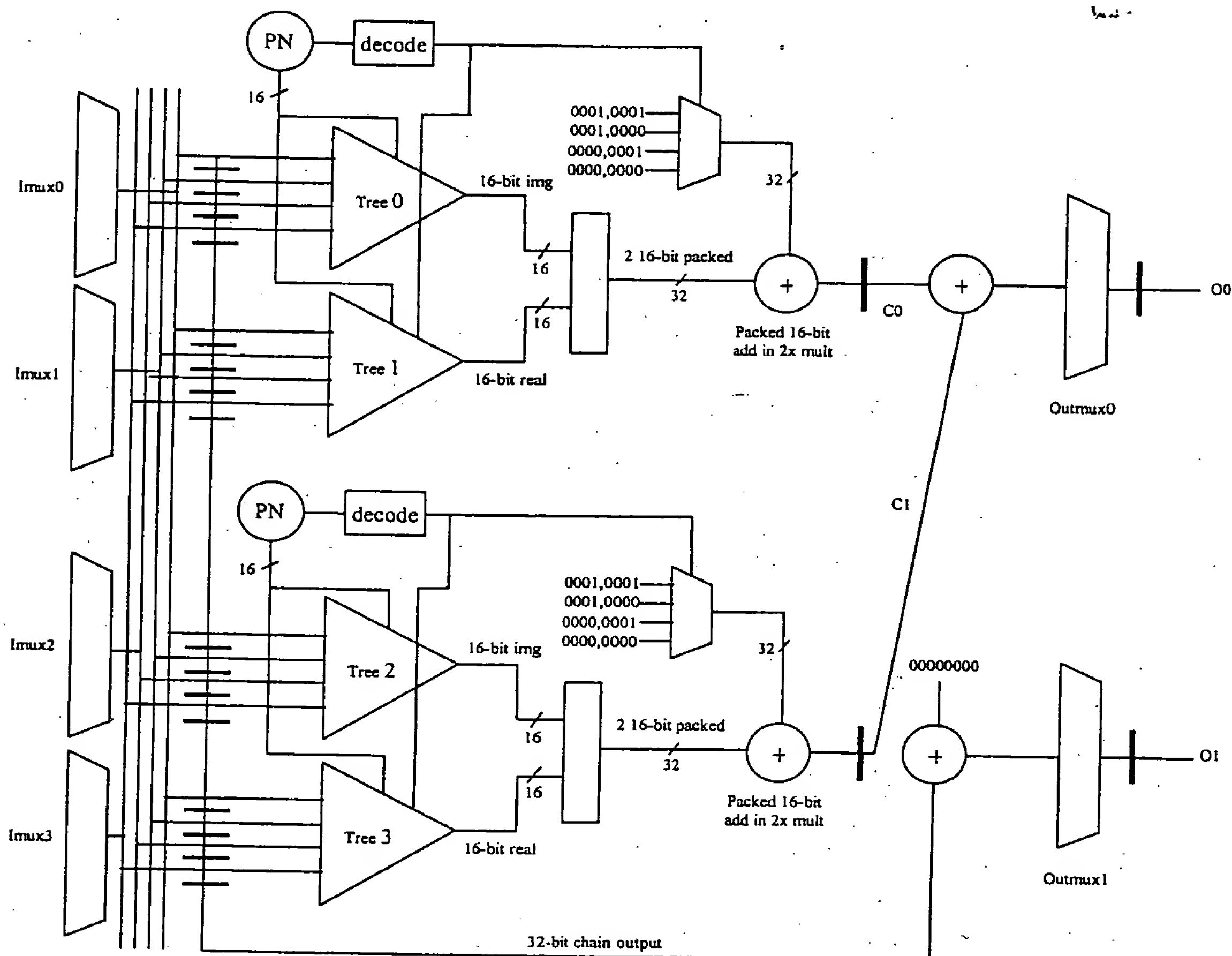
16 rows at 10 um each??

Despreader integration with input and Output muxes



- 16-bit output of each 'real' despreader tree is packed with the corresponding 'imaginary' despreader output into one 32-bit output, such as output of Tree0 is packed with output of Tree1 and Tree 2's is packed with Tree3's.
- The final add before the output mux is performed inside the 2x multiplier in 4ADD16(packed 16-bit addition) mode.
- A add-one signal decoded inside the despreader is used to determine the other operand of the final add. The operand could either be zero or 2 packed 16-bit '0001'.

Correlator integration with input and Output muxes



- 32-bit chain output is added with all zero in the 2x mult before being sent to output mux 1.
- 2 32-bit packed outputs C0 and C1 are added together before being sent to output mux 0.

A 5-bit opcode is used in the enhanced multiplier (both 2xmuilt and desp/corr) for decoding 9 modes in 2xmuilt and 12 mode desp/corr as shown in the following table.

| Mode | Bit[4] | Bit[3] | Bit[2] | Bit[1] | Bit[0] |
|-----------------|--------|--------|--------|--------|--------|
| 4xdesp8 complex | 1 | 1 | 1 | 0 | 0 |

| | | | | | |
|---------------------|---|---|---|---|---|
| 4xdesp8 comp-conjug | 1 | 1 | 1 | 0 | 1 |
| 4xdesp8 zero | 1 | 1 | 1 | 1 | 0 |
| 4xdesp8 real | 1 | 1 | 1 | 1 | 1 |
| 8xdesp8 complex | 1 | 1 | 0 | 0 | 0 |
| 8xdesp8 comp-conjug | 1 | 1 | 0 | 0 | 1 |
| 8xdesp8 zero | 1 | 1 | 0 | 1 | 0 |
| 8xdesp8 real | 1 | 1 | 0 | 1 | 1 |
| Corr complex | 1 | 0 | 1 | 0 | 0 |
| Corr comp-conjug | 1 | 0 | 1 | 0 | 1 |
| Corr zero | 1 | 0 | 1 | 1 | 0 |
| Corr real | 1 | 0 | 1 | 1 | 1 |
| 2MULT | 0 | 0 | 0 | 0 | 0 |
| 4ADD32 | 0 | 0 | 1 | 0 | 0 |
| 4ADD16 | 0 | 0 | 1 | 0 | 1 |
| 4MULT | 0 | 1 | 0 | 0 | 0 |
| 4MULTSUM | 0 | 1 | 0 | 0 | 1 |
| 4MULT2SUM | 0 | 1 | 0 | 1 | 0 |
| 4FIR | 0 | 1 | 1 | 0 | 0 |
| CMULT | 0 | 1 | 1 | 1 | 0 |
| CMULT16 | 0 | 1 | 1 | 1 | 1 |

- There is an output register in each of the desp/corr tree.
- The pn code will be registered after the 2-to-1 input scrambling mux.
- All desp/corr trees output would go to an adder in the 2xmilt before going to the output mux.